UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.
09/078,933	05/14/98	BLANDY G	AT9-98-071

TM02/0813

DUKE E YEE
P O BOX 802334
DALLAS TX 75380

EXAMINER	
DAS,C	
ART UNIT	**PAPER NUMBER**
2122	16

DATE MAILED: 08/13/01

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner of Patents and Trademarks

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
ASSISTANT SECRETARY AND COMMISSIONER OF
PATENTS AND TRADEMARKS
Washington, D.C. 20231

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Paper No. 16

Application Number: 09/078,933
Filing Date: 5/14/98
Appellant(s): Blandy

Duke W. Yee

For Appellant

EXAMINER'S ANSWER

This is in response to appellant's brief on appeal filed on 7/9/01.

(1) Real Party Interest

A statement identifying the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

A statement identifying the related appeals and interferences which will directly affect or be directly affected by or have a bearing on the decision in the pending appeal is contained in the brief.

(3) Status of Claims

The statement of the status of the claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Invention

The summary of invention contained in the brief is correct.

(6) Issues

The appellant's statement of the issues in the brief is correct.

(7) Grouping of Claims

The appellant's statement the grouping of claims is correct.

(8) Claims Appealed

The copy of the appealed claims contained in the Appendix to the brief is correct.

(9) Prior Art of Record

The following is a listing of the prior art of record upon in the rejection of claims under appeal.

5,784,553	Kolawa et al	7-1998

(10) New Prior Art

No new prior art has been applied in this examiner's answer.

(11) Allowable Subject Matter

There is no allowable subject matter in this application.

(12) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

1. Claims 1-32 are rejected under 35 U.S.C 102(a) as being anticipated by Kowala el al, US Patent no.5,784,553

Claim Rejections - 35 USC § 102

2. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless --

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

3. Claims 1-32 are rejected under 35 U.S.C. 102(a) as being anticipated by Kolawa et al, US Patent No. 5,784,553.

As per claim 1, Kolawa teaches **identifying a path within the method that is being executed** is shown in column 11 line 42-44 ("the dynamic symbolic execution is performed along **the path taken by an actual execution of the program**"), path taken by an actual execution of the program inherently including identifying a path within a the method that is being executed, **plurality of instructions are associated with the paths** is shown in column 24 line 30-32("The TGS Driver Program executes **the program for the path** that corresponds to the input initially

chosen and, **for all the instructions found in that path"),** translating the first type instruction for the path being executed is shown in column 3 line 34-42 (" the computer program **being represented by JAVA bytecodes after being compiled by a JAVA compiler** . The method includes the steps of reading the JAVA bytecodes obtaining an input value for the computer program; **symbolically executing an instruction of the computer program represented in the JAVA bytecodes"**), compiling **JAVA bytecode inherently including instructions are translated into second type instruction as claimed, unexecuted path remain untranslated** is shown in column 5 line 53-55 ("The original source code 11 comprises all types of **files used to express an uncompiled, that is, non- object code** , computer program, including definitional and declarative files"), **uncompiled, that is, non- object code** inherently including unexecuted path remain in translated as claimed, **the path is one of the path** is shown in column 6 line 39-42 ("Each program is broken down into a series of code blocks comprising one or more program statements occurring along a single path of execution"), plurality of paths is shown in column 2 line 42-44 (" every branch at least once and optionally for generating as many paths as desired in the total path coverage set").

As per claim 2, 5, 14, 18, 27, 32, Kolawa et al teach executing second type instruction for a path in response to a loop back through the path is shown in column 22 line 56-61 ("The SVM then gets the next JAVA bytecode from the .class files (block 427). If all JAVA program instructions have not been symbolically executed (block 428), control returns to the top of the

control loop (block 421) for processing of the next program instruction"), during execution of the routing is shown in column 26 line 41-43 (" A normal JAVA Virtual Machine can be used to execute the instrumented bytecodes").

As per claim 3, 6, 13, 19, 26, 31, Kolawa et al teach **translated instruction for the path are executed in an order** is shown in column 25 line 60-64 ("the TGS Driver Program reads the JAVA Bytecodes in the .class files for the module under test at step 480. The Module Testing Module 238 at step 482 decides on a **first sequence of methods to be executed**"), **sequence of methods to be executed** inherently including translated instructions are stored in an execution order.

As per claim 4, 17, Kolawa et al teach **identifying a path within the method that is being executed** is shown in column 11 line 42-44 ("the dynamic symbolic execution is performed along **the path taken by an actual execution of the program**"), path taken by an actual execution of the program inherently including identifying a path within a the method that is being executed, **bytecodes are associated with the paths** is shown in column 21 line 18-21 (" The SVM symbolically interprets the JAVA program under test using information stored in the JAVA **Bytecodes 210**. Upon completion, **symbolic expressions and associated path conditions** are output by the SVM of the bytecodes are a relatively high-level representation of the source code so that some optimization and machine code generation (via a just -in-time compiler 214) can be performed") at that level all required branch conditions (block 302)**", compiling the byte**

codes is shown in ABSTRACT line 3-5 ("The JAVA program comprises program statements and program variables represented as JAVA source code and compiled by a **JAVA compiler into JAVA bytecodes,")**, byte codes are compiled into native machine code is shown in column 17 line 51-54 ("The bytecodes are a relatively high-level representation of the source code **so that some optimization and machine code generation (via a just-in-time compiler 214) can be performed** at that level"), Kowala teaches bytecodes **remain uncompiled** is shown in column 5 line 53-55 ("The original source code 11 comprises all types of **files used to express an uncompiled,** that is, non- object code , computer program, including definitional and declarative files"), **uncompiled files** inherently including unexecuted path as claimed, **the path is one of the path** is shown in column 6 line 39-42 ("Each program is broken down into a series of code blocks comprising one or more program statements occurring along a single path of execution"), plurality of paths is shown in column 2 line 42-44 (" every branch at least once and optionally for generating as many paths as desired in the total path coverage set").

As per claim 7, Kolawa et al teach **JIT in compiling method** is shown in column 17 line 62-65 (" optionally turned into machine code by the **Just-in-time Compiler 214.** The JAVA interpreter and Just-In-Time Compiler operate in the context of a Runtime System 222").

As per claim 8, 15, 21, 28, Kolawa et al teach **data structure** is used during compiling of the method store information about a path as claimed is shown in column 6 line 37-42 ("Fig. 2D is a **data structure for storing block and branch analysis information** extracted from the

original computer program. Each program is broken into a series of code blocks comprising one or more program statements occurring along a single path of execution").

As per claim 9, 22, Kolawa et al teach data structure stores the native machine codes is shown in column 6 line 37-42 and column 17 line 51-54 ("Fig. 2D is a **data structure for storing block and branch analysis information** extracted from the original computer program") and ("The bytecodes are a relatively high-level representation of the source code **so that some optimization and machine code generation (via just in time compiler 214) can be performed** at that level").

As per claim 10, 20, 23, Kolawa et al teach data structure is JIT station is shown in column 5 line 52-54 (""Fig. 2D is a **data structure for storing block and branch analysis information** extracted from the original computer program.), storing block and branch analysis information inherently including JIT compiler information as claimed.

As per claim 11, 24, 30, Kolawa et al teach identifying the method that is to be executed is shown in column 20 line 36-38 ("Module 41 **identifies the branch condition controlling the execution of that code block"), compiling the byte codes** is shown in column 26 line 44-47 ("FIG 23 is a flow diagram of the steps for handling **JAVA bytecode instrumentation**. At step 492, the JAVA Parser 230 reads the .class file 210 generated by the **JAVA Compiler** 208 and the .Java files 200 of the original source code").

As per claim 12, 25, Kolawa et al teach unexecuted paths remain in a bytecode form is shown in column 5 line 53-55 ("The original source code 11 comprises all types of **files used to express an uncompiled, that is, non- object code** , computer program, including definitional and declarative files"), **uncompiled, that is, non- object code** inherently including unexecuted path remain in translated as claimed.

As per claim 16, 29, Kolawa et al teach **data structure stores the instruction** is shown in column 4 line 36-38 ("1G.2D is **a data structure for block and branch analysis information stored** in the program database of FIG. 2A")

(13) Arguments

The applicant has argued:

(A) Kolawa et al do not teach the identifying a path in the routine being executed.

(B) Kolawa does not provide any teaching or suggestion of translating the instructions from one type to another for a path being executed.

(C) Kowala does not teach storing the compiled instructions in an execution order.

(D) Kowala does not teach the uncompiled byte codes.

(E) Kowala does not teach executing instructions native machine code for a path within a plurality of paths.

(F) Kowala does not teach translated instructions for the path are executed in an order.

(G) Kowala does not teach a data structure is used during compiling of the method.

(H) Kowala does not teach data structure stores the native machine code.

(I) Kowala does not teach JIT in compiling method.

(J) Kowala does not teach compile the bytecode.

Response to Arguments

(A) Kowala teaches identifying a path in the routine being executed is shown in column 24 line 32 ("The TGS Driver Program executes the program for the path that corresponds to the input initially chosen and, for all of the instructions found in that path"), choosing the path inherently including identifying a path as claimed.

(B) Kowala teaches translating the instructions from one type to another for a path being executed is shown in column 28 line 40-44 (" the step of transforming further comprising the steps of: selecting one of the program statements in the JAVA bytecodes; symbolically executing the selected program statement"), and column 3 line 34-42, transforming the Java source code to byte code inherently including translating the instruction from one type to another type as claimed.

(C) Kowala teaches storing the compiled instructions is an execution order is shown in column 3 line 54-57 (" The method includes the steps of reading the JAVA bytecodes for the module; **determining a sequence of JAVA methods to be called in the module** and arguments to be passed to the JAVA methods; performing symbolic interpretation of the **sequence of JAVA methods of the JAVA bytecodes** by calculating symbolic expressions for the JAVA methods"),

sequence of JAVA methods of the JAVA bytecodes inherently including compiled instruction is an execution order.

(D) Kowala teaches unexecuted paths remain untranslated is shown in column 5 line 53-55 ("The original source code 11 comprises all types of files used to express an uncompiled, that is, non object code"), uncompiled, that is, non object code inherently including unexecuted path remain untranslated as claimed.

(E) Kowala teaches executing native instruction is shown in column 3 line 35-43.

Respectfully submitted,

(F) Kolawa et al teach **translated instruction for the path are executed in an order** is shown in column 25 line 60-64 ("the TGS Driver Program reads the JAVA Bytecodes in the .class files for the module under test at step 480. The Module Testing Module 238 at step 482 decides on a **first sequence of methods to be executed**"), **sequence of methods to be executed** inherently including translated instructions are stored in an execution order.

(G) Kolawa et al teach **data structure** is used during compiling of the method store information about a path as claimed is shown in column 6 line 37-42 ("Fig. 2D is a **data structure for storing block and branch analysis information** extracted from the original computer program. Each program is broken into a series of code blocks comprising one or more program statements occurring along a single path of execution").

(H) Kolawa et al teach data structure stores the native machine codes is shown in column 6 line 37-42 and column 17 line 51-54 ("Fig. 2D is a **data structure for storing block and branch analysis information** extracted from the original computer program") and ("The bytecodes are a relatively high-level representation of the source code **so that some optimization and machine code generation (via just in time compiler 214) can be performed** at that level").

(I) Kolawa et al teach **JIT in compiling method** is shown in column 17 line 62-65 (" optionally turned into machine code by the **Just-in-time Compiler 214.** The JAVA interpreter and Just-In-Time Compiler operate in the context of a Runtime System 222").

(J) Kolawa et al teach compile the bytecode is shown in column 26 line 66-67.

Respectfully submitted

Chameli C. Das

8/8/01

Chameli C. Das

MARK R. POWELL
SUPERVISORY PATENT EXAMINER
GROUP 2700

Conferee

Todd Ingberg